Chapitre 3

LANGAGES ET GRAMMAIRES

1. INTRODUCTION

- Au-delà de la classe des langages rationnels, il existe bien d'autres familles de langages, qui valent également la peine d'être explorées.

- Les langages réguliers s'expriment à l'aide d'ERs. Mais la plupart du temps, les langages ne sont pas réguliers. On a besoin d'un outil plus puissant que les ERs : les grammaires.

- Tout langage de programmation possède des règles qui indiquent la structure syntaxique d'un programme bien formé.

- La syntaxe d'un langage peut être décrite par une grammaire.

- Chomsky identifie une hiérarchie de familles de grammaires de complexité croissante, chaque famille correspondant à une contrainte particulière sur la forme des règles de récriture.

- Nous nous intéressons, ici, en particulier aux grammaires hors contextes.

2. GRAMMAIRES

Exemple 1: Dans le langage naturel, une phrase est composée d'un sujet d'un verbe suivi d'un complément :

L'étudiant subit un cours

PHRASE = SUJET VERBE COMPLÉMENT
SUJET = ARTICLE ADJECTIF NOM |
ARTICLE NOM ADJECTIF | ARTICLE NOM
ARTICLE = le | la | un | des | l'
ADJECTIF = malin |stupide | COULEUR
COULEUR = vert | rouge | jaune
...

Exemple 2: Une expression conditionnelle en C:

if (EXPRESSION**)** INSTRUCTION

Il faut définir ce qu'est une EXPRESSION et ce qu'est une INSTRUCTION

- Une grammaire est un ensemble de règles permettant de dire si un mot c.-à-d. une suite de symboles est correcte ou non, comme dire aussi si une phrase c.-à-d. une suite de mots est correcte ou non.

- Une grammaire est la donnée de $\mathbf{G=(V_T, V_N, S, P)}$ où

  - V_T est un ensemble non vide de symboles **terminaux** (alphabet terminal, Ex : le, la, if,…)

  - V_N est un ensemble non vide de symboles **non-terminaux** avec $VT \cap VN = \varnothing$ (symboles qu'il faut encore définir, Ex : PHRASE, SUJET, ARTICLE, COULEUR,...)

  - S est un symbole initial $\in$ VN appelé **axiome** (Ex. : PHRASE)

  - P est un ensemble de **règles de production** (règles de réécriture)

- Une règle de production $\alpha \rightarrow \beta$ précise que la séquence de symboles α : partie gauche de la production ($\alpha \in \mathbf{(V_T U V_N)^+}$) peut être remplacée par la séquence de symboles β : partie droite de la production ($\beta \in \mathbf{(V_T U V_N)^*}$).

Exemple 3:

Symboles terminaux (alphabet) : $V_T = \{a,b\}$ Symboles non terminaux : $V_N = \{S\}$
Axiome : S
Règles de production :

$$\left\{ \begin{array}{l} S \rightarrow \varepsilon \\ S \rightarrow aS \end{array} \right. \Leftrightarrow \quad S \rightarrow \varepsilon \mid aSb$$

Exemple 4:

G=(V_T, V_N, S, P) avec
V_T={il, elle, parle, est, devient, court, reste, sympa, vite}
V_N={PHRASE, PRONOM, VERBRE, COMPLEMENT, VERBETAT, VERBACTION}
S=PHRASE
P={ PHRASE → PRONOM VERBE COMPLEMENT PRONOM
→ il | elle
VERBE → VERBETAT | VERBACTION
VERBETAT → est | devient | reste
VERBACTION → parle | court
COMPLEMENT → sympa | vite }

NB:

- S. non-terminaux : Lettres capitales (A, B,…,Z)
- S. terminaux : Lettres minuscules du début de l'alphabet (a,b,…),
- Chaîne de S. terminaux : Lettres minuscules de la fin de l'alphabet (t,…,z),
- Chaîne de S. terminaux et non-terminaux :Lettres grecques (α, β, …)

3. ARBRE DE DÉRIVATION

- On appelle **dérivation** l'application d'une ou plusieurs règles à partir d'un mot de $(V_T U V_N)^+$
- → dérivation obtenue par application d'une seule règle de production
- $\overset{*}{\longrightarrow}$ dérivation obtenue par application de *n* règles de production avec $n \geq 0$

Exemple 1: Sur la grammaire de l'exemple 1

$$\begin{cases} S \rightarrow \varepsilon \\ S \rightarrow aSb \end{cases}$$

aSb → aaSbb

S →* ab S→aSb→ab

S →* aaabbb S→aSb→aaSbb→aaaSbbb→aaabbb

Exemple 2: Sur la grammaire de l'exemple 2

PHRASE → PRONOM VERBE COMPLEMENT

PHRASE $\overset{*}{\longrightarrow}$ elle VERBETAT sympa

PHRASE $\overset{*}{\longrightarrow}$ elle parle vite

PHRASE $\overset{*}{\longrightarrow}$ elle court sympa

NB:
On peut générer des phrases syntaxiquement correctes mais qui n'ont pas de sens. C'est l'analyse sémantique qui permettra d'éliminer ce problème.

- Étant donné une grammaire G, on **note L(G) le langage généré par G** et défini par $\{w \in (V_T)^* / S \rightarrow^* w\}$

Exemple 1:

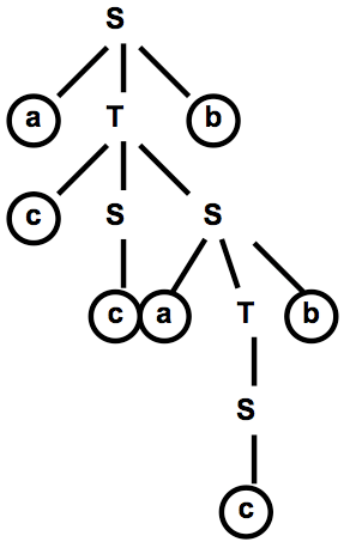
La grammaire de l'exemple 1 nous donne $L(G)=\{a^n b^n, n\geq 0\}=\{\varepsilon, ab, aabb, aaabbb,\ldots\}$

- On appelle **arbre de dérivation (arbre syntaxique)** tout arbre tel que
  - La racine est l'axiome,
  - Les nœuds sont les symboles non terminaux,
  - Les fils d'un nœud α sont $\beta_0,\ldots, \beta_n$ ssi $\alpha \rightarrow \beta_0,\ldots, \beta_n$ est une règle de production

Exemple : Soit la grammaire suivante :

$$P=\begin{cases} S\rightarrow aTb \mid c \\ T\rightarrow cSS \mid S \end{cases}$$

Un arbre de dérivation pour le mot *accacbb* est :



- **Dérivations gauches** : réécrite le symbole non-terminal le plus à gauche à chaque étape

  S⟶aTb⟶acSSb⟶accSb⟶accaTbb⟶accaSbb⟶accacbb

- **Dérivations droites** : réécrite le symbole non-terminal le plus à droite à chaque étape

  S⟶aTb⟶acSSb⟶acSaTbb⟶acSaSbb⟶acSacbb⟶accacbb

Ces deux suites différentes de dérivations donnent le même arbre de dérivation.

- Une grammaire est dite **ambiguë** s'il existe un mot de L(G) ayant plusieurs arbres syntaxiques.

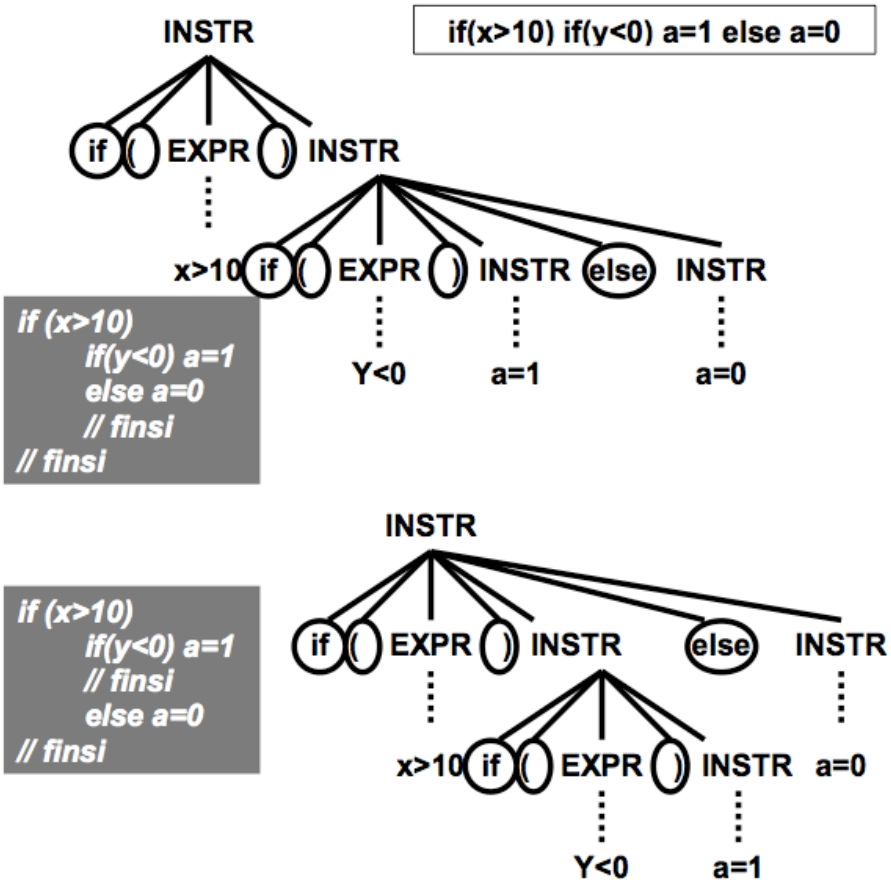
Exemple :

Soit G donnée par

INSTR →if (EXPR) INSTR else INSTR
INSTR →if (EXPR) INSTR
INSTR →…
EXPR →…

Cette grammaire est ambiguë car le mot m :

possède deux arbres syntaxiques différents (avec les mêmes feuilles dans le même ordre).



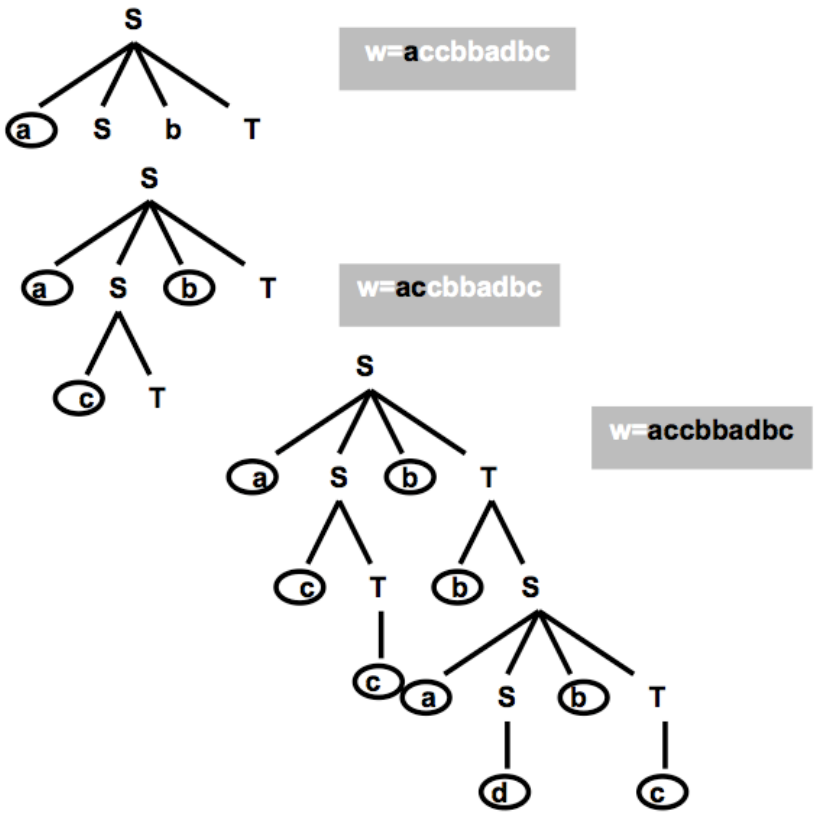
4. ANALYSE DESCENDANTE

Principe : construire l'arbre de dérivation du haut (la racine : axiome de départ) vers le bas (les feuilles : ULs).

Exemple 1 :

S→aSbT|cT|d
T→aT|bS|c

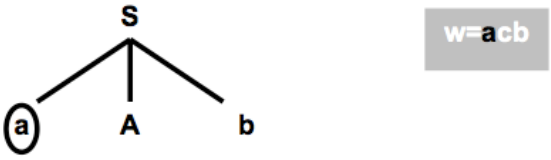
avec le mot w=**accbbadbc**



Chaque règle commence par un terminal différent → On sait immédiatement laquelle prendre.

Exemple 2 :

S→aAb
A→cd|c avec le mot w=**acb**



En lisant le **c**, on ne sait pas s'il faut prendre la règle **A→cd** ou **A→c .**

→ 2 possibilités :

- Lire aussi la lettre suivante : **b**
- Donner la possibilité de faire des retours en arrière.

Exemple 3 :

S→aSb | aSc |d avec le mot w=aaaaaaadbbcbbbc

Pour savoir quelle règle utiliser, il faut connaître aussi la dernière lettre du mot .

Exemple 4 :

E→TE'
E'→+TE' |-TE'|ε
T→ FT' avec le mot w=3*4+10*(5+11)/34+12
T'→ *FT'| /FT'|ε
F→ (E)|nb

→ Avoir une table qui nous dit : quand je lis tel caractère et que j'en suis à dériver tel symbole non-terminal, alors j'applique telle règle : **Table d'analyse**

4.1. TABLE D'ANALYSE LL(1)

- Pour construire une table d'analyse, on a besoin des ensembles **PREMIER** et **SUIVANT.**

4.1.1. Calcul de PREMIER

∀α , terminal ou non-terminal, PREMIER(α)={a, a∈VT / ∃α →* aβ}

1) Si **X est un non-terminal** et **X → $Y_1Y_2…Y_n$** est une production avec Yi symbole terminal ou non_terminal, alors

a) ajouter les éléments du PREMIER(Y_1) sauf ε dans PREMIER(X)

b) si **∃ j (j ∈ {2,…,n})/ ∀ i=1…j-1** on a ε∈ **PREMIER(Y_i)**, alors ajouter les éléments de PREMIER(Yj) sauf ε dans PREMIER(X)

c) si **∀ i=1…n** ε ∈ **PREMIER(Y_i)**, alors ajouter ε dans PREMIER(X)

2) Si **X est un non_terminal** et **X→ε** est une production, alors ajouter ε dans PREMIER(X)

3) Si **X est un terminal** alors PREMIER (X)={X}

Recommencer jusqu'à ce qu'on n'ajoute rien de nouveau dans les ensembles PREMIER

Exemple :

E→TE'	**PREMIER(E)=PREMIER(T)={(,nb}**	1)a)
E'→+TE' | -TE' | ε	**PREMIER(E')={+,-,ε}**	1)a)-3)-2)
T →FT'	**PREMIER(T)=PREMIER(F)={(,nb}**	1)a)
T'→ *FT' | /FT' | ε	**PREMIER(T')={*,/,ε}**	1)a)-3)-2)
F → (E) | nb	**PREMIER(F)={(,nb}** 1)a)	

4.1.2. Calcul de SUIVANT

∀A , non-terminal, SUIVANT(A)={a, a∈VT / S →* αAaβ}

1) Ajouter $ à SUIVANT(S) où **S est l'axiome** de départ de la grammaire

2) ∀ **A→ α B** β où **B est un non terminal**, ajouter PREMIER(β) à SUIVANT(B), sauf ε

3) ∀ **A→ α B**, ajouter SUIVANT(A) à SUIVANT(B)

4) ∀ **A→ α B** β avec ε∈**PREMIER(β)**, alors ajouter SUIVANT(A) à suivant (B)

Recommencer à partir de l'étape 3) jusqu'à ce qu'on n'ajoute rien de nouveau dans les ensembles SUIVANT

Exemple :

a) E→TE'	**SUIVANT(E)={$,)}**	1)a)-2)e)
b) E'→+TE' | -TE' | ε	**SUIVANT(E')={$,)}**	3)a)b)
c) T →FT'	**SUIVANT(T)={+,-,), $}**	2)b)-4)b)
d) T'→ * FT' | /FT' | ε	**SUIVANT(T')={+,-,), $}**	3)c)
e) F → (E) | nb	**SUIVANT(F)={*,/,),+,-,$}**	2)d)-3)c)

4.1.3. Construction de la table LL

Pour chaque production **A→α** faire

1) ∀ **a ∈ PREMIER(α) (a≠ε),** rajouter A→α dans la case M[A,a]

2) si ε∈**PREMIER(α)**, alors ∀ b ∈SUIVANT(A), ajouter A→α dans M[A,b]

Chaque case M[A,a] vide est une erreur de syntaxe

Exemple :

PRODUCTIONS	PREMIERS	SUIVANTS
E→TE'	**{(,nb}**	**{$,)}**
E'→+TE' | -TE' | ε	**{+,-,ε}**	**{$,)}**
T →FT'	**{(,nb}**	**{+,-,), $}**
T'→ *FT' | /FT' | ε	**{*,/,** ε**}**	**{+,-,), $}**
F → (E) | nb	**{(,nb}**	**{*,/,,),+,-,$}**

	nb	+	-	*	/	(	$	)
E	E→TE'					E→TE'		
E'		E'→+TE'	E'→- TE'				E'→ε	E'→ε
T	T→FT'					T→FT'		
T'		T'→ε	T'→ε	T'→*FT'	T'→/FT'		T'→ε	T'→ε
F	F→nb					F→(E)		

4.2. ANALYSEUR SYNTAXIQUE

Algorithme

Données : mot *m* terminé par $, table d'analyse *M*
Résultat : S−∗→m ?

Initialisation de la pile P: pointeur *ps* sur la 1ère lettre de *m*
S
$

Répéter
X : sommet de P
a : lettre pointée par *ps*
si *X* est un non-terminal **alors**
si $M[X,a]=X\rightarrow Y_1 \dots Y_n$ **alors**
enlever *X* de *P*
mettre Y_n puis $Y_{n-1} \dots Y_1$ dans *P*
émettre en sortie la production $X\rightarrow Y_1 \dots Y_n$
sinon case vide dans la table
Finsi

Sinon
Si *X=$* **alors**
Si *a=$* **alors** ACCEPTER **Sinon** ERREUR **Finsi**
sinon
Si *X=a* **alors** enlever *X* de *P*
Avancer ps
Sinon ERREUR
Finsi
Finsi
Finsi
Jusqu'à ERREUR ou ACCEPTER

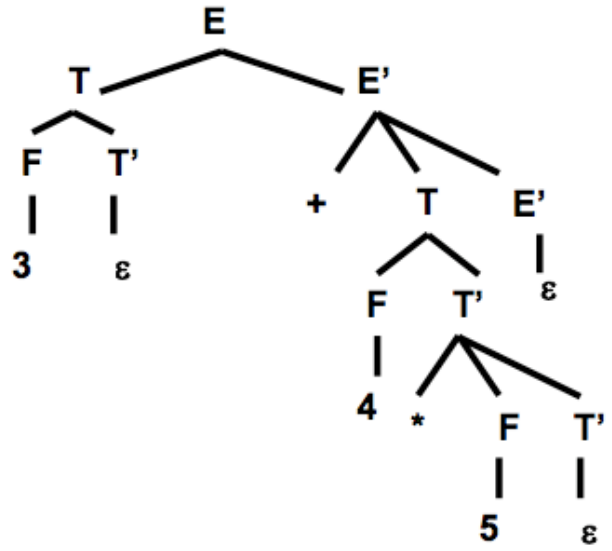
Exemple : m=3+4*5

E→TE' **T→FT'** **F→ (E) | nb**

E'→+TE' | -TE' | ε **T'→ FT' | /FT' | ε**

P	Entrée	Sortie
$E	**3+4*5$**	**E→TE'** (enlever E, mettre E'T)
$E'T	**3+4*5$**	**T→FT'** (enlever T, mettre T'F)
$E'T'F	**3+4*5$**	**F→nb** (enlever F, mettre 3)
$E'T'3	**3+4*5$**	(enlever 3, avancer)
$E'T'	**+4*5$**	**T'→ε** (enlever T', mettre ε)
$E'	**+4*5$**	**E'→+TE'** (enlever E', mettre E'T+)
$E'T+	**+4*5$**	(enlever +, avancer)
$E'T	**4*5$**	**T→FT'** (enlever T, mettre T'F)
$E'T'F	**4*5$**	**F→nb** (enlever F, mettre 4)
$E'T'4	**4*5$**	(enlever 4, avancer)
$E'T'	***5$**	**T'→*FT'** (enlever T', mettre T'F*)
$E'T'F*	***5$**	(enlever *, avancer)
$E'T'F	**5$**	**F→nb** (enlever F, mettre 5)
$E'T'5	**5$**	(enlever 5, avancer)
$E'T'	**$**	**T'→ε** (enlever T', mettre ε)
$E'	**$**	**E'→ε** (enlever E', mettre ε)
$	**$**	**ACCEPTER** (analyse syntaxique réussie)

Arbre syntaxique :



4.3. GRAMMAIRE LL(1)

- On appelle **grammaire LL(1)** une grammaire pour laquelle la table d'analyse décrite précédemment n'a aucune case définie de façon multiple.

- LL(1) :
  - L (Left to Right scanning) : parcours d'entrée de gauche à droite
  - L (Left most derivation) : utilisation des dérivations à gauche
  - 1 : un seul symbole de prévision est nécessaire pour la prise d'une décision d'action d'analyse

Exemple 1 : Grammaire non factorisée à gauche

$\begin{cases} S \rightarrow aAb \\ A \rightarrow cd \mid c \end{cases}$

PREMIER(S)={a} SUIVANT(S)={$}

PREMIER(A)={c} SUIVANT(A)={b}

	a	c
S	S → aAb	
A		A→cd A→c

<u>Exemple 2</u> : Grammaire récursive à gauche

$$\begin{cases} S \rightarrow aTbbbb \\ T \rightarrow Tb \mid \varepsilon \end{cases}$$

- Une grammaire **ambiguë** ou **récursive à gauche** ou **non factorisée** n'est pas LL(1)

4.4. RÉCURSIVITÉ À GAUCHE

- Une grammaire est **immédiatement récursive à gauche** si elle contient un non-terminal A / $\exists$ **A→A** α où α est une chaîne quelconque.

Exemple : Grammaire

$$\begin{cases} S \rightarrow ScA \mid B \\ A \rightarrow Aa \mid \varepsilon \\ B \rightarrow Bb \mid d \mid e \end{cases}$$

- **Élimination de la récursivité à gauche immédiate :**

Remplacer toute règle de la forme **A→A** $\alpha \mid \beta$ par

- **A→** β **A'**
- **A'→** α **A'** $\mid \varepsilon$

La grammaire ainsi obtenue reconnaît le même langage que la grammaire initiale.

Exemple : Grammaire'

$$\begin{cases} S \rightarrow BS' \\ S' \rightarrow cAS' \mid \varepsilon \\ A \rightarrow \varepsilon A' \\ A' \rightarrow a\ A' \mid \varepsilon \\ B \rightarrow dB' \mid eB' \\ B' \rightarrow bB' \mid \varepsilon \end{cases}$$

$A \rightarrow aA \mid \varepsilon$

S→ Sα $\mid \varepsilon \Leftrightarrow$ **S→** α**S** $\mid \varepsilon$
on peut se passer de A'

- Mot : **dbbcaa** :

- Grammaire :

S→ScA→BcA→BbcA→BbbcA→dbbcAc→ dbbcAaa→dbbcaa

- Grammaire':

S→BS'→dB'S'→dbB'S'→dbbB'S'→dbbS'→dbbS'→dbbcAS'→dbbcA'S'→dbbcaaA'

S'→dbbcaaS' →dbbcaa

- Une grammaire est **récursive à gauche** si elle contient un non-terminal A / $\exists$ **A**$\overset{+}{\rightarrow}$**A** α où α est une chaîne quelconque.

Exemple :

$$\begin{cases} S \rightarrow Aa \mid b & \text{S : n'est pas immédiatement récursif à} \\ A \rightarrow Ac \mid Sd \mid c & \text{gauche mais récursif à gauche : } \mathbf{S \rightarrow Aa \rightarrow Sda} \end{cases}$$

- **Élimination de la récursivité à gauche :**

Ordonner les non-terminaux $A_1, A_2, \ldots, A_n$
Pour i=1 **à** n **faire**
  Pour j=1 **à** i-1 **faire**
    Remplacer $\mathbf{A_i \rightarrow A_j\alpha}$ **où** $\mathbf{A_j \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_p}$ par
    $\mathbf{A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_p\alpha}$
  Finpour
  Éliminer les récursivités à gauche immédiates des productions A_i
Finpour

La grammaire ainsi obtenue reconnaît le même langage que la grammaire initiale.

Exemple :

$$\begin{cases} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid c \end{cases}$$

On ordonne S, A
i=1 pas de récursivité immédiate dans S→Aa | b
i=2 et j=1 on obtient A→ Ac | Aad | bd| c
  A→ bdA' | cA'
  A'→ cA' | adA'| ε

Grammaire obtenue :

$$\begin{cases} S \rightarrow Aa \mid b \\ A \rightarrow bdA' \mid cA' \\ A' \rightarrow cA' \mid adA' \mid \varepsilon \end{cases}$$

Exemple :

$$\begin{cases} S \rightarrow Sa \mid TSc \mid d \\ T \rightarrow TbT \mid \varepsilon \end{cases} \Rightarrow \begin{cases} S \rightarrow TScS' \mid dS' \\ S' \rightarrow aS' \mid \varepsilon \\ T \rightarrow T' \\ T' \rightarrow bTT' \mid \varepsilon \end{cases}$$

Or on'a $S \rightarrow TScS' \rightarrow T'ScS' \rightarrow ScS'$ (encore récursif à gauche) $\Rightarrow$
l'algorithme ne marche pas toujours lorsque on'a une règle : $T \rightarrow \varepsilon$

5.5. GRAMMAIRE PROPRE

- Une grammaire est dite **propre** si elle ne contient aucune production $\mathbf{A} \rightarrow \varepsilon$
- **Rendre une grammaire propre :** remplacer A par ε dans toute production dans laquelle A apparaît dans sa partie droite

Exemple :

$$\begin{cases} S \rightarrow aTb \mid aU \\ T \rightarrow bTaTA \mid \varepsilon \\ U \rightarrow aU \mid b \end{cases} \Rightarrow \begin{cases} S \rightarrow aTb \mid ab \mid aU \\ T \rightarrow bTaTA \mid baTA \mid bTaA \mid baA \\ U \rightarrow aU \mid b \end{cases}$$

5.6. FACTORISATION À GAUCHE

Exemple :

$$\begin{cases} S \rightarrow \mathbf{aEbS} \mid \mathbf{aEbS}eB \mid a \\ E \rightarrow \mathbf{bc}B \mid \mathbf{bc}a \\ B \rightarrow ba \end{cases}$$

- L'idée : pour développer un non-terminal A quand il n'est pas évident de choisir quelle production prendre, on doit réécrire les productions de façon à différer la décision jusqu'à ce que suffisamment de texte ait été lu pour faire le bon choix.

- **Factorisation à gauche :**

  - Pour chaque **non-terminal A**, trouver le plus long préfixe α commun à deux de ses alternatives ou plus.
  - Si $\alpha \neq \varepsilon$ remplacer **A**→ $\alpha\beta$**1**|…|$\alpha\beta$**n** |λ**1**|…|λ**p** (où les λi ne commencent pas par α) par

$$A \rightarrow \alpha A' | \lambda_1 | \ldots | \lambda_p$$

$$A' \rightarrow \beta_1 | \ldots | \beta_p$$

  - Recommencer jusqu'à ne plus en trouver

$$\begin{cases} S \rightarrow \mathbf{aEbS} \mid \mathbf{aEbS}eB|a \\ E \rightarrow \mathbf{bc}B \mid \mathbf{bc}a \\ B \rightarrow ba \end{cases} \Rightarrow \begin{cases} S \rightarrow \mathbf{aEbS}S'|a \\ S' \rightarrow eB|\varepsilon \\ E \rightarrow \mathbf{bc}E' \\ E' \rightarrow B \mid a \\ B \rightarrow ba \end{cases}$$

<u>Conclusion</u> :

• Si la grammaire est LL(1), l'analyse syntaxique peut se faire par l'analyse descendante.

• Étant donné une grammaire :

1) La rendre non ambiguë
2) Éliminer la récursivité à gauche si nécessaire
3) La factoriser à gauche si nécessaire
4) Construire la table d'analyse
5) Espérer que ça soit LL(1)

- Contre exemple :

La grammaire suivante n'est pas LL(1) or elle n'est pas récursive à gauche, elle est factorisée à gauche et elle n'est pas ambiguë :

$$\begin{cases} S \rightarrow aTb|\varepsilon \\ T \rightarrow cSa|d \end{cases}$$

- **Remarque :** Il y a un autre type d'analyse qui est l'analyse ascendante. Cette méthode permet d'analyser plus de grammaires que la méthode descendante (car il y a plus de grammaires LR que LL(1)). Dans cette méthode, il n'a strictement aucune importance que la grammaire soit récursive à gauche.